Review of Content Languages Suitable for Agent-Agent Communication[1]

Luis Botelho
luis.botelho@iscte.pt
ADETTI

Steven Willmott
steven.willmott@epfl.ch
EPFL

Tianning Zhang
zhang@agentscape.de
Agentscape

Jonathan Dale
jonathan.dale@fla.fujitsu.com
Fujitsu Laboratories of America

Abstract

This technical report provides an evaluation of several possible languages and semantic formalisms that could be used in agent communication to play the role of a content language in the EU Agentcities.RTD project. The conclusions and background information may however be useful for agent developers more generally.

This document includes a description of candidate languages, a list of criteria applied, evaluations of the five candidate languages and a final evaluation. The five candidate languages were DAML+OIL, ebXML, FIPA-SL, KIF and Prolog and the choice made for the EU Agentcities.RTD project was to develop services in KIF, FIPA-SL or both. Furthermore it is expected that the number and type of content language used in the EU Agentcities.RTD project will evolve over time as tests are carried out.

The review process which led to the authoring of this document was carried out in the context of the Agentcities.RTD IST funded project (IST-2000-28385) and we would like to thank all project partners who contributed to it. The opinions expressed in this paper are those of the authors and are not necessarily those of the EU Agentcities.RTD partners.

[1] Version 2.0, 25 May, 2002.

1 Introduction

Within an agent cooperation framework, the content language provides a key layer for exchanging data and knowledge among agents. Message contents coded in the content languages are interpreted by the agents within the context and constraints specified by other message components. Content processing offers the major mechanism for flexible and intelligent agent interactions based on exchanging and sharing of domain-specific, application-relevant agent knowledge.

Theoretically, many programming and knowledge representation languages can be used to code the agent communication contents. However, different languages, with their specific theoretical bases and targeted application areas, have heterogeneous levels of expressiveness, complexity, and available support from the agent community or development/deployment platforms.

Generally speaking, an expressive content language can help to enhance the flexibility of agent cooperation relations by encoding wider range of relevant information and knowledge. On the other hand, the complexity of implementation and understanding, which can be associated to an expressive language, can significantly influence the acceptability and usability of the solution in some application contexts. Moreover, support from popular platforms and major agent communities can be also an important factor in determining the development cost, and the possible penetration/acceptance of an agent-based application.

This technical report aims at

- Identifying the key criteria for the suitability of a language for representing agent communication content, and
- Evaluation of the suitability of a number of key candidate languages.

For this purpose, this report derives the criteria from FIPA standardization efforts and the applications in the Agentcities.RTD project. Evaluation focuses on some selected candidates that have currently significant influence in agent and semantic web applications.

The five candidate languages were DAML+OIL, ebXML, FIPA-SL, KIF and Prolog and the choice made for the EU Agentcities.RTD project was to develop services in KIF, FIPA-SL or both. Furthermore it is expected that the number and type of content language used in the EU Agentcities.RTD project will evolve over time as tests are carried out.

Comment: This is the same as the abstract; we should write a separate introduction that provide a little more motivation.

2 Role of the Content Language

Communication between software systems is often characterized as a number of levels to separate different aspects of communication. Table 1 provides such a level decomposition drawn from those often used for FIPA and KQML/KIF.

Level	Description	Semantic Description
Context	State of the world in which the *conversation* takes place	Formalism for describing the meaning of states of the world, an institution, a market, etc.
Conversation	Sequence of *messages*	Formal account of the meaning of statements in the protocol description formalism – which can ideally be interpreted to give the meaning of any particular state in the conversation sequence (AUML, FSM etc.)
Message	A single communication from one or more originators to one or more listeners that expresses the speaker's opinion about the *content*[2]	Formal account of the meaning of messages represented in a particular language, for example: • FIPA-ACL semantics in Modal logic • KQML semantics in Definite Clause Grammar formalism • ebXML message semantics in natural language
Content	The description of a partial world state (or a world) which may contain references to *objects*, *actions*, *functions*, … in one or more *domains*	Formal grammar, semantics represented in particular language and a definition of those semantics, for example: • FIPA-SL: logic base • KIF: logic base • Prolog: logic base + interpreter • Java: language + JVM
Domain Description	References to and definite descriptions of *objects, action, function* and other instances	Formalism for defining possible classes and/or instances of things in the world, for example, DAML+OIL and Ontolingua KIF.

Table 1: Semantic Communication Stack[3]

A domain description may be arbitrary types according to how its description is formalized, for example, in DAML+OIL [DAML+OIL] everything is a subtype of the Thing class, but every class defined has its own identifier. Content expressions would normally be expressed in a content language (such as KIF [KIF], FIPA-SL [FIPA00008]), the message in an agent communication language (such as FIPA-ACL [FIPA00061], KQML [KQML]) and the conversation/context perhaps in a logical formalism such as situation calculus with the protocol sequence specified in AUML [AUML], for example.

[2] The simplest and most usual case would be one receiver and one sender, but with more powerful semantic formalisms it could be more.

[3] Note that this breakdown differs from the Semantic Web stack [].

The content language therefore expresses views of the world related to a particular communication an agent makes and often references instances or descriptions of objects or other entities that are externally defined in an ontology.

3 Candidate Content Languages

A large number of formalisms could be used as content languages from imperative programming languages to declarative modal logics. For this review, we therefore address only the following five candidates that were proposed by EU Agentcities.RTD project partners:

- **DAML+OIL**: An RDF and description logic-based formalism originally intended for expressing ontologies.
- **ebXML**: A framework intended for communication between business systems.
- **FIPA-SL**: A content language developed by FIPA and often used in conjunction with FIPA-ACL.
- **KIF**: A knowledge representation language that is used as an interchange format between knowledge systems and often used in conjunction with KQML.
- **Prolog**: A logic programming language.

4 Evaluation Criteria and Project Requirements

The content language evaluation criteria that have been applied to the candidate languages are grouped into areas described in the following sections.

4.1 Expressivity Requirements

Content language Expressivity means the amount and complexity of natural language sentences (or concepts) that may be expressed using the content language. In general, the more expressive the language, the more difficult it is to build computational mechanisms to process it (Note however that tractability is not directly considered in this review).

Independently of the complexity of the application domain, if a content language is to be used with FIPA-ACL communication language, it must satisfy three requirements:

1. It must be capable of representing propositions,
2. It must be capable of representing actions (not their semantics, only their designators), and,
3. It must be capable of representing objects, including identifying referential expressions[4] to describe objects.

Languages that cannot be used to express the above types of concepts cannot be used with the full range of FIPA-ACL performatives. In particular:

- Inform messages require propositions as content:

```
(inform :content "((is-blue car))")
```

- Request messages require action expressions as content:

```
(request :content "((action you make-tea))")
```

- Query messages require object references as content:

```
(query-ref :content "((all ?x (is-red ?x)))")
```

A content language does not need to be very expressive if any of the following two conditions hold true:

1. Where agents have rigid interfaces and cannot or do not need to deal with more complex expressions that are built from simpler ones. This is the case for applications in which agents do not require more than API-like interfaces.

[4] Used to represent the open questions *what*, *which* and *who*.

2. Where provider agents have only rigid, simple information processing capabilities and where requesting agents with complex information need to use simple information requests and are responsible for all subsequent information integration. This approach may be used in many application domains, but it generally leads to severe inefficiencies. Firstly, all information processing is centralised in the demanding agent and secondly, large unrestricted amounts of data must be conveyed from information providers to information requesters, for example, where the requester asks for two large tables of the provider database in order perform a join which returns only a few records.

In the EU Agentcities.RTD project, neither of the above conditions hold true so the content language needs to be an expressive language. The capability to express rich propositions is the most demanding requirement on EU Agentcities.RTD project content language choice since propositions are used for many purposes within the message contents, such as, in questions, in assertions, in reasons (for instance when a proposal is rejected), and, to express conditions for the execution of requested actions.

Besides atomic propositions, it is likely that any content language choice will need to support the expression of:

- Propositions with explicit or implicit quantification,
- Propositions with logical connectives (not, and, or, implies and equivalence),
- Modal propositions (believe, intend, desire), and,
- Action propositions, such as propositions that describe states of the world in which a particular action has been executed and states of the world in which it would be possible to execute a certain action.

Finally, the language should be expressive enough to refer to objects, actions and propositions from arbitrary application domains, that is, anything we can write an ontology for.

4.1.1 Examples

Ex1: Action propositions with modal operators

The Lisbon event planner would like to know if the pop artist Prince is going to perform a show in Barcelona in order to see if it could be cheaper to hold it in Lisbon. Prince is represented by the SexyMF agent who can express the question "do you intend to perform in Barcelona?" as follows:

```
Intends (SexyMF, Done (BarcelonaFShow))
```

`BarcelonaFShow` is an action designator which can be more complex.

Ex2: Quantifiers plus logical connectives

The Lisbon event planner would like to know the names of all theatres with more than 200 seats where no seat is closer to the stage than 20 meters. The ontology has the following predicates and functions:

`Theater/1`	A predicate that maintains the names of theatre
`Seat/2`	A predicate that relates a theatre with each of its seats
`Number-of-Seats/1`	A function that takes the name of a theatre and returns its number of seats
`Distance-to-stage/2`	A function that takes a seat and a theatre name and returns the distance of that seat to the theatre's stage

```
All (t, Theatre (t) ∧ Number-of-Seats (t) ≥ 2000 ∧ ∀s [Seat (t, s) ⇒
  Distance-to-Stage (s, t) ≥ 20])
```

4.1.2 Expressivity Test

Since some of the languages being considered are not logic-based, the following table of example expressions is used to identify what can be expressed in each language.

Expression	Language Representation	Details
"Schrödinger's Cat is alive"	Proposition about a particular instance	N/A
"Cats are animals"	Proposition about a class of things	
"You making the tea"	An action expression (reference to an action instance)	
"Drinking too much is bad for you"	Proposition about a class or type of actions	
"All red things"	Direct reference to things (objects)	
"Any colour a car might have"	Direct reference to values of certain object properties	
"All things are hot"	Universal quantification	
"Something is cold"	Existential quantification	
"Younger than 8 or older than 60"	Disjunction	
"The desired movie should be romantic and the cinema should at walk distance"	Conjunction	
"The transport should not be private"	Negation	
"Success implies Payment"	Implication	
"Luis has the persistent goal that W"	Persistent goal modal operator	
"Steve Believes X"	Belief modal operator	
"Jonathan Desires Y"	Desire modal operator	
"Matthias Intends Z"	Intention modal operator	

4.2 Parsing and Processing Tools

This refers to the available processing tools including parsers, language translators and reasoners, such as theorem provers, inference engines, and planners. Parsers may not be necessary if the message contents are serialised objects of the agent implementation language, but they are necessary whenever the content language is encoded into a string format, for example, XML and s-expressions.

Parsers are the minimum requirement, while inference and other reasoning mechanisms are important but not a requirement since not all agents require a reasoning mechanism.

4.2.1 Evaluation

Considerations for these tools are:

1. **Availability**: Are tools available for the target language?
2. **License**: Commercial, open source or free?
3. **Integration**: Are the tools easy to integrate with existing agent platforms since parsers will exist between the transport layer and the agent, that is, it is in the data path?

Language translators are used when it is convenient to convert some message content into another language that may easily be used by the agent. Translators may also be needed to convert expressions of an internal content language into expressions of an external content language. Consider an information agent that stores information in a relational database which uses SQL and an internal content language. If this agent receives a query that is expressed in FIPA-SL, then it will need to convert the FIPA-SL content expression into SQL commands in order to be able to execute the query.

The term reasoner is used when referring to any reasoning mechanism, from theorem provers to planners to inference engines. Reasoners have to be used in a variety of situations in inter-agent communication and the most common are cases in which the agent, which is using the same representation language as the content language, receives a question. Sometimes the answer to this

question is explicitly represented in the knowledge base of the agent, but often, the agent must perform some amount of reasoning in order to be able to answer a question that is not explicitly represented in its knowledge base.

4.2.2 Examples

Knowledge base:

```
{Citizen (Jonathan),
 Lives (Steve, Switzerland), ∀p ∀c Lives (p, c) ⇒ Citizen (p)}
```

Query: `Tell me the names of all known citizens`

Answer: `Jonathan, Steve`

While `Jonathan` is explicitly represented to be a `Citizen`, the agent performed some inference to conclude that `Steve` is also a `Citizen`.

4.2.3 Notes

It is not mandatory that the agent is a knowledge-based system. Even if the knowledge is implicitly encoded into the program data structures and procedures of the agent, there still needs to be some procedures to derive not-encoded knowledge from previously encoded knowledge.

It is also important to evaluate the commercial availability of these tools.

4.3 Knowledge Acquisition Facilities

Knowledge acquisition facilities are tools to help knowledge-based system engineers to create their knowledge bases. Examples include:

- **Editors**: Graphical or other tools for manipulating or describing message instances.
- **Validators**: Checking the correctness of messages.
- **Compilers and import features**: Mechanisms for transforming human readable or editor output into agent code.

Content languages do not have to be knowledge representation languages. It is possible to have a knowledge-based agent with a one knowledge representation language (for example, OPS5, Prolog, Golden Works or KEE) and a different content language. However, content languages have similar properties to knowledge representation tools and therefore it also makes sense to have tools that facilitate the task of writing message contents. These tools might include specialised editors (for syntax enforcing), and tools to help writing content expressions by examples.

It is also important to evaluate the commercial availability of these tools.

4.4 Language Learning Difficulty

This is an estimate of the time it would take for a person without previous knowledge related to the language would take to learn it and use practical examples:

1. **Agent Developers**[5]: The learning curve is towards the ability to develop required services.
2. **Researchers**: The learning curve is in understanding examples and the ability to apply it to their own problems.
3. **Industry**: The learning curve is in understanding examples and the ability to apply it to their problems.
4. **Lay people**: The learning curve is in understanding examples.

4.5 Migration Paths and Flexibility

This is an evaluation of the degree of flexibility offered by a particular content language:

[5] In particular, people already familiar with agents, the FIPA agent standard and FIPA agent platforms.

1. **Syntax**: Is it possible to migrate to new syntaxes?
2. **Semantics**: Is it possible to add user defined constructs? Is it possible to generalise or restrict meanings to make the language more powerful?
3. **Combination**: Is it possible to embed fragments of other languages in messages?
4. **Implementation**: Are tool implementations for the language reusable for other languages or other tasks?

4.6 Adoption Issues

This is a subjective assessment of the benefits of each choice considering resulting impact on:

- Likely adoption and interworking with other research projects,
- Likely adoption and existing user communities in industry, and,
- Ability to influence standards bodies and the support a language already has.

4.7 Summary and Relative Importance

The first two criteria in the following list have more importance when evaluating a particular content language that the others, which are considered to have approximately equal importance:

1. Expressivity (with the caveat that not everything may be needed initially),
2. Availability of message parsing and processing tools,
3. Availability of knowledge acquisition/manipulation facilities,
4. Difficulty of learning the learning,
5. Migration paths and flexibility, and,
6. Adoption issues.

The following table presents the list of all attributes that must be considered in the content language evaluation.

Expressivity			
01. Represents propositions	Yes / No		
02. Represents actions	{Atomic, Composed} / No		
03. Represents open questions	Yes / No		
04. Allows quantification	Explicit / Implicit / No		
05. Allows connectives	{AND, OR, NOT, EQUIV, IMPLIES} / No		
06. Allows action propositions	{Done, Feasible} / No		
07. Allows modal operators	{I, B, G, PG, U} / No		
08. Allows functions	Yes / No		
Available Processing Tools	**Tool Name**	**License/Price**	**Comments**
09. Parsers			
10. Translators			
11. Reasoners			
Knowledge Acquisition Facilities	**Tool Name**	**License/Price**	**Comments**
12. Editors			
13. Others			
Language Learning Difficulty	1 (Very Easy) – 5 (Very Difficult)		
14. Agent Developers			
15. Researchers			
16. Industry			
17. Lay people			
Migration Paths			
18. Evaluation			
Adoption Issues	1 (Low Preference) - 3 (High Preference)		
19. Research preference			
20. Industry preference			
21. Standards impact			

5 DAML+OIL

5.1 Basis of the Review

The basis of this review is the DAML+OIL specification [DAML+OIL].

5.2 Expressivity

DAML+OIL is a language for specifying ontologies. In particular it is intended for making statements about:

- Types and classes of objects and entities which exist in the world,
- Instances of such types and classes which exist in the world, and,
- The properties of such types and classes and instances.

The semantics for DAML+OIL are defined for statements such as class definitions, instance definitions, property definitions and restrictions on classes, instances and property values. It is therefore clear that DAML+OIL would be a very effective tool for expressing such ontological knowledge in agent communication, but it is not clear how useful it would be for expressing other types of communication.

The following two observations are important to make about DAML+OIL:

- Everything in DAML+OIL is an ontology. That is, DAML+OIL files usually begin with the declaration of an ontology and it is not clear if definitions are valid outside the context of such a declaration. It might be assumed, however, that something like "Message" could also be defined.
- All statements are propositions. In particular, they are propositions about the existence of classes or instances or about the properties which apply to classes or instances. For example, the following states that the are in the world of this ontology things called cats that are a type of animal:

```
<daml:Class rdf:ID="cat">
  <rdfs:subClassOf rdf:resource="#animal">
</daml>
```

  Additionally, the following states that there is in the world defined by this ontology an instance of a cat called Nermal which is alive (one can now also infer that there exists at least one animal in the world):

```
<Cat rdf:ID= "Nermal">
  <status>alive</status>
</Cat>
```

5.2.1 Expressivity Test

Expression	Language Representation	Details

"Schrödinger's Cat is alive"	`<Cat rdf:ID= "schrödinger-s_cat">`
	`  <owner>Shrodinger</owner>`
	`  <status> alive </status>`
	`</Cat>`
		This is not quite the same since it states that there exists in the world a cat called Schrodinger's cat which is alive[6].

"Cats are animals"	`<daml:Class rdf:ID="cat">`
	`  <rdfs:subClassOf`
	`  rdf:resource="#animal">`
	`</daml>`
		This is the kind of statement for which DAML+OIL is intended.

"You making the tea"	`<making_tea rdf:ID= "instance_1">`
	`  <actor>You</actor>`
	`</making_tea>`
		As with the first example, this states that there exists an instance

[6] Note that the statement is therefore acting like a constructor rather than a reference. While this is close, it would cause problems in many systems where it is important to be able to make direct references to things.

		of an action called making_tea in which You are the actor.
"Drinking too much is bad for you"	`<daml:Class rdf:ID="excessive_drinking"> <rdfs:subClassOf rdf:resource="#bad_things"/> </daml>`	There are multiple ways to express this. Another way can be to use properties.
"All red things"		As with the first example, it can be said that there exist red things or that all things are red but they cannot be referenced.
"Any colour a car might have"		This can be expressed closely by defining a property car colour with the domain "cars" and then listing all of the colours. Again this is not a direct reference.
"All things are hot"	`<daml:Class rdf:about "#Thing"> <rdfs:subClassOf> <daml:restriction daml:hasValue="hot"> <daml:onProperty rdf:resource= "temperature"/> </daml:restriction> </rdfs:subClassOf> <daml:Class>`	The Thing class is the top level entity in the DAML world so every sub class can be constrained to be "hot".
"Something is cold"	`<thing rdf:ID= "cold_thing"> <temperature>cold</temperature> </thing>`	
"Herring or Perch"	`Disjunction`	See Implication.
"Vodka and Tonic"	`Conjunction`	See Implication.
"Not cricket"	`Negation`	See Implication.
"Success implies Payment"	`<daml:Class rdf:ID="Implication"> … </daml>` `<Implication ref:Class= "logic_1"> <precedent>Success</precedent> <antecedent>Payment</antecedent> </Implication>`	This would rely on defining a class of implications and then making the statement that there exists an implication of a particular thing.

"Luis has the persistent goal that W"	`<PersistentGoal rdf:ID= "Wgoal"> <owner> luis </owner> <content>W</content> </PersistentGoal>`
	or
	`<Person rdf:ID= "Luis"> <PersistentGoals> W </PersistentGoals> </Person>`
		Again it can only be stated that something exists. So for persistent goals, it is possible to say that there is a persistent goal. Alternatively one could embed it into the description of `Luis`.

"Steve Believes X"		See Persistent Goals.
"Jonathan Desires Y"		See Persistent Goals.
"Matthias Intends Z"		See Persistent Goals.

5.2.2 Summary

Attempting to express the statements in the table above in DAML+OIL is an interesting exercise and reveals the following things:

- DAML+OIL is good for expressing certain types of information, such as class declarations, and instance declarations, but its semantics do not go far beyond this. It is a good way to define languages (as a meta-languages) but not really sufficiently general in its own right. If it were to be used as the basis for a new language, then this would need to be a language defined in

DAML+OIL with its own semantics for the new terms defined, restrictions on how the components can be composed, etc.

- It does not allow direct references to entities (objects or actions) and instead statements have to be made that such a thing exists in the world, which is not the same.
- It could be possible to define any type of operator which may exist in the world including modal operators, although when creating an instance of the operator one is force to revert to 'there exists…" Note that while this is very useful, it leads to using a content language wrapped in DAML+OIL (with its own, new, semantics) and not expressed in DAML+OIL itself.
- It is not clear how defined logical operators could be conveniently composed (because of the implicit existential quantification).

As a final note in this area, frameworks such as Sesame [Sesame] and Algae [Algae] are developing RDF query engines that may make it possible to generate object references.

5.3 Parsing and Processing Tools

There are many tools available for DAML+OIL[7]. Furthermore, RDF and RDFS tools can be used to support at least some DAML+OIL functionality, but not all of the language.

Name	Details
Jena [Jena]	This has an internal RDF model and a specific API to load DAML+OIL files, to manipulate elements, to change values, etc. It has a HP-specific license which appears to allow re-use and derived software as long as the authors name is not used to endorse the resulting product.
RedLand [RedLand]	An RDF application framework which allows the manipulation of RDF triples, objects, etc., but it appears not to have DAML specific support. The license is LGPL or MPL.
Wilbur RDF Toolkit [Wilbur]	A toolkit for RDF and DAML+OIL which includes a DAML+OIL parser as an extension of the RDF parser. The license is the Nokia Open Source License.
ATOMIK [ATOMIK]	A toolkit for ontology and agent language manipulation including support for FIPA-SL, FIPA-ACL (s-expressions and XML), KQML, FIPA-KIF and DAML+OIL ontologies. The license is LGPL.

There is a DAML API that has been developed for manipulating DAML objects in Java [DAML-API] which can provide part of the parsing solution. Standard RDF parsers, Jena and ATOMIK appear to be relatively easy to integrate into Java agent platforms. The RDF basis means that there are at least some tools available for other platforms, for example, Wilbur for LISP.

5.4 Knowledge Acquisition Facilities

The majority of tools for DAML are for manipulation or reasoning with ontologies:

- **Viewing**: 4 Browsers and 2 viewers (1 for the Palm).
- **Generation**: 2 Crawlers.
- **Translators**: 1 XSLT adapter, PDDL to DAML, XMLSchema to DAML.
- **Validation**: 1 validator and 3 analysers.
- **Editors**: 3 editors[8] (DUET, OILed and ONTOedit).

5.5 Language Learning Difficulty

This is dependent on the expressivity evaluation and can be summarised as follows:

[7] See http://www.daml.org/

[8] Protégé does not current support DAML+OIL.

- In areas of functionality for which DAML+OIL was designed, it should be relatively easy to learn. It has simple object oriented or declarative structures as well as a number of viewers and editors that support learning. Furthermore, validators should take some of the difficulty out of learning and applying the language by pointing out errors.
- In areas of functionality for which DAML+OIL was not designed and which require either additional usage guidelines or language definitions, the functionality of the new language would determine how quickly it could be learned. One immediate impact would be the lack of tool support for the extensions and another would be the need to use the two specifications simultaneously.

Besides these considerations is the structure and syntax of RDF which is arguably harder to understand than either XML or a simple logical framework (s-expressions). Regarding the different user groups:

- **Agent developers**: Would probably find standard DAML+OIL fine to use but if extensions/new language definitions are proposed then the ease of use depends upon the nature of those extensions.
- **Researchers**: Would probably find standard DAML+OIL fine to use but if extensions are proposed, then the ease of use depends upon the nature of those extensions.
- **Industry**: Would probably find DAML+OIL accessible through the tools that are provided but it is not clear what impact additional rules would have.
- **Lay people**: Standard DAML+OIL captures concepts that can be easily presented to lay people, although probably in an abstract form.

5.6 Migration Paths and Flexibility

This is divided into two areas:

- **Syntax**: Being based on an RDF/XML syntax is a clear advantage since a very large number of ontology initiatives and business standards use XML based syntaxes.
- **Semantics**: DAML+OIL has a semantics that is limited to its intended domain of discourse which does not limit extensions in principle and also provides very little structural support for additions or extensions.

While DAML+OIL has some advantages here, the semantics are potentially more important than the syntax in terms of migration paths (since it is a relatively simple matter to retrofit a different syntax onto an existing language).

5.7 Adoption Issues

DAML+OIL and the Semantic Web activities are receiving interest fundamentally for their usage in ontology specification and *not necessarily* as a content language.

5.8 Summary

Expressivity	
Represents propositions	Yes
Represents actions	No
Represents open questions	No
Allows quantification	Implicit
Allows connectives	No
Allows action propositions	No
Allows modal operators	No
Allows functions	No (however, a function may be represented by a relation)

Available Processing Tools	**Tool Name**	**License/Price**	**Comments**
Parsers	Jena, Redland, Wilbur, ATOMIK	Free, assorted licenses but some are LGPL (which should be acceptable)	Good support here and long term view to efficiency since it is based on RDF
Translators	Translators from XMLSchema, PDDL + a generic XSLT adaptor	Free, license details not found	The XSLT adaptor in particular could be very useful
Reasoners	Euler, Triple, cmw	Not clear from the sites but probably free	Very early stage work in all three cases
Knowledge Acquisition Facilities	**Tool Name**	**License/Price**	**Comments**
Editors	Duet, ONTOedit, OILed	Free, license less relevant since there is no need to bundle	Some are not complete or stable
Others	An array of other tools which may come in handy		
Language Learning Difficulty	1 (Very Easy) – 5 (Very Difficult)		
Agent Developers	2 (4)[9]		
Researchers	2 (5)		
Industry	2 (5)		
Lay people	2 (5)		
Migration Paths			
Evaluation	Syntax migration and integration is strong since it is based on RDF/XML, but the semantic framework is not very general which makes it unclear how to migrate.		
Adoption Issues	1 (Low Preference) - 3 (High Preference)		
Research preference	1 (Since it would require new structures to be added)		
Industry preference	2 (RDF/XML/DAML+OIL definitions would help, but additional structures may be unpopular)		
Standards impact	3 (Would be high if an easy way to use DAML+OIL as a content language were developed)		

[9] Figures in parentheses are for extensions to DAML+OIL.

6 ebXML Review

6.1 Basis of the Review

The basis of this review is on the ebXML specification [ebXML].

6.2 Expressivity

ebXML is not defined as a general purpose knowledge representation language, as a result, it is certainly less expressive in general purpose knowledge acquisition. On the other hand, ebXML is strongly related to EDI, designed by keeping in mind the special requirements for modelling business processes and collaborations, and shall be more suitable for such applications. The key expressivity features can be summarized in the following:

- Basically, ebXML does not support the direct representation of propositions about the states/properties of an agent (like the propositions in a FIPA inform message). Propositions, for example about the capability/states of the submitting party are implied by the information submitted to the ebXML registry.
- Actions are represented in ebXML as business transactions or activities.
- The relationship between concepts or object classes and object instances is specified in ebXML via the classification of objects to a hierarchy of classification nodes, that is, concepts.
- Queries related to object references are realized by the ebXML query mechanisms supported via the ebXML registry services.
- Quantifiers are not supported explicitly. The simple deployment of a universal quantifier can be implicitly represented via the classification hierarchy, for example, all cats are animals.
- Modal operators are implied by the registry operation and business collaboration activities, and are not specified in the language itself.

6.2.1 Expressivity Test

Expression	Language Representation	Details
"Schrödinger's Cat is alive"	<ClassificationNode id="livingThingNode" name="livingThing"/> <ClassificationNode id="catXNode" name="catX" parent="catNode"/> <ClassificationNode id="SchrödingerNode" name="Schrödinger"/> <Classification id="statusClassification" classifiedObject="catXNode" classificationNode="LivingThingNode"/> <Classification id="ownerClassification" classifiedObject="catXNode" classificationNode="SchrödingerNode"/>	
"Cats are animals"	<ClassificationNode id="animalNode" name="animal"/> <ClassificationNode id="catNode" name="cat" parent="animalNode"/>	This is used for hierarchical classification of services.

"You making the tea"	<BusinessTransaction
	  name="Make Tea">
	  <RequestingBusinessActivity name=""
	  <DocumentEnvelope
	  businessDocument="businessDoc"
	  <DocumentEnvelope/>
	  </RequestingBusinessActivity>
	</BusinessTransaction>
		An actor can only request actions from other parties via proposing a collaboration protocol agreement.

"Drinking too much is bad for you"		ebXML does not support general logical statements.

"All red things"	<RegistryEntryQuery>
	<hasClassificationBranch>
	<ClassificationNodeFilter>
	  <Clause>
	  <SimpleClause leftArgument = "name">
	  <StringClause
	  stringPredicate="equal">
	  "red"
	  </StringClause>
	  </SimpleClause>
	 </Clause>
	</ClassificationNodeFilter>
	</hasClassificationBranch>
	</RegistryEntryQuery>
		This is used to query the registry for all objects that are classified as "red". Thus, if the corresponding registry/classification actions were submitted this would work.

"Any colour a car might have"	<ClassificationNodeQuery>
	  <PermitsClassificationBranch>
	  <RegistryEntryQuery>
	  <hasClassificationBranch>
	  <Clause>
	  <SimpleClause leftArgument =
	  "description">
	  <StringClause
	  stringPredicate=
	  "contains">
	  "car"
	  </StringClause>
	  </SimpleClause>
	  </Clause>
	  </hasClassificationBranch>
	  </RegistryEntryQuery>
	  </PermitsClassificationBranch>
	
	 <HasParentNode>
	 <ClassificationNodeFilter>
	  <Clause>
	  <SimpleClause leftArgument = "name">
	  <StringClause
	  stringPredicate="equal">
	  "colour"
	  </StringClause>
	  </SimpleClause>
	 </Clause>
	 </ClassificationNodeFilter>
	 </HasParentNode>
	</ClassificationNodeQuery>
		This is used to query the registry about any colour nodes that are used to classify any "car" nodes. It is assumed that the description of such node contains the key word "car".

"All things are hot"	<Classification
	  id="property"
	  classifiedObject="thingNode"
	  classificationNode="hotNode"/>	

"Something is cold"		
"Herring or Perch"		`or` can only be used to query the registry.
"Vodka and Tonic"		`and` can only be used to query the registry.
"Not cricket"		Negation is not supported.
"Success implies Payment"		Implication (and the associated reasoning) is not supported.

"Luis has the persistent goal that W"		A goal is represented by a collaboration protocol profile, which specifies the services and the protocols supported by a business party. By registering such a CPP in the registry, the owner of the CPP commits and publishes its business goal, intension and desires to the business community.
"Steve Believes X"		Definition of beliefs is not supported.
"Jonathan Desires Y"		See Persistent Goals.
"Matthias Intends Z"		See Persistent Goals.

6.3 Parsing and Processing Tools

There are currently few commercial and open source platforms that support the core components in an ebXML-based business co-operation environment. As ebXML message syntaxes are defined in XML schema or DTD, it is possible to deploy any XML parser with schema/DTD validation capability to parse the messages. Semantic-related validations, however, must be realized at a higher layer within the ebXML business co-operation platform or platform components.

Name	Details
Open ebXML	There are a number of open source ebXML tools and components[10]: • **Binary Mark-up Language**: A faster and more compact representation of XML. • **Workbench**: A GUI workbench for editing process definitions, viewing message stores, etc. • **Red-Line**: A Business process Server. • **Registry**: An open implementation of the ebXML registry. • **Message handler**: A high performance message handler for ebXML messages. • **SHS**: Open source implementation of the Swedish governmental protocol SHS. • **Tools**: A set of tools to use when working with ebXML. • Pretty printing of Collaboration-Protocol Profile XML files using XSLT.
ebXML Registry and Repository	This is a free Java technology-based the Sun ebXML Registry and Repository Implementation. This package can be used out of the box to submit, store, retrieve and manage XML resources based on the ebXML Registry Information Model 1.0 and the ebXML Registry Services Specification 1.0. This latest Registry/Repository implementation includes enhanced support for form-based authentication and ebXML query/retrieval methods.
JAXR	JAXR provides an API for a set of distributed Registry Services that enable B2B integration between business enterprises, using the ebXML protocols.
Component-X	This platform provides a simple and standards-based approach to

[10] See http://www.ebxml.org/implementations/index.htm

	assembling XML-Java components for Web Services, B2B, Enterprise Integration and supply chain automation that supports ebXML.

6.4 Knowledge Acquisition Facilities

Open ebXML is hosting two projects for delivering tools supporting knowledge acquisition, aiming at implementing:

- A GUI workbench for editing process definitions, viewing message stores etc., and ,
- A set of tools to use when working with ebXML such as pretty printing of Collaboration-Protocol Profile XML files using XSLT.

6.5 Language Learning Difficulty

ebXML is not a traditional knowledge representation framework and has therefore not such a strong basis in predicate logics. On the other hand, it is strongly related to traditional frameworks like EDI for the business process management and coordination. Therefore, knowledge acquisition based on ebXML requires less expertise in logics and knowledge engineering, but more expertise in the context of business process engineering.

- **Agent developers, researchers and lay people**: It is relatively difficult since it is based on many concepts for commercial business transactions and processing management, which can be non-intuitive for research communities from other context.
- **Industry**: Adoption is easy since it uses concepts that are common in commercial business transaction and processing management.

6.6 Migration Paths and Flexibility

This can be analysed in the following aspects:

1. **Syntax**: Using XML technology, ebXML can be easily extended with new syntactical elements to support new functional features.
2. **Semantics**: By adding new elements with new semantics, ebXML can be extended to modelling and managing real business processes and business co-operations.
3. **Combination** The definition of ebXML framework allows the utilization of specifications from other languages under certain standard conformance conditions.
4. **Implementation**: Tools for registry management and for process management and coordination can be reused in other business cooperation environments.

6.7 Adoption Issues

ebXML is based on the tradition of EDI technology, which has wide applications in the industry which is a factor contributing to the acceptance of the ebXML framework.

6.8 Summary

The following table presents the list of all attributes that must be considered in the content language evaluation.

Expressivity	
01. Represents propositions	Yes (to some extent)
02. Represents actions	Yes
03. Represents open questions	Yes (only to the registry)
04. Allows quantification	Implicit (Partially)
05. Allows connectives	Yes (only and/or and only to the registry)
06. Allows action propositions	No
07. Allows modal operators	Implicit
08. Allows functions	No

Available Processing Tools	**Tool Name**	**License/Price**	**Comments**
09. Parsers	any XML parsers	Free licenses	
10. Translators	Open ebXML	Open source	Not yet available
11. Reasoners	Open ebXML, Sun ebXML Registry, X-component, JAXR	Free evaluation. Open Source (Open ebXML)	Some are not yet available until early 2002
Knowledge Acquisition Facilities	**Tool Name**	**License/Price**	**Comments**
12. Editors	Open ebXML	Open source	Not yet available
13. Others			
Language Learning Difficulty	1 (Very Easy) – 5 (Very Difficult)		
14. Agent developers	3		
15. Researchers	4		
16. Industry	1		
17. Lay people	3		
Migration Paths			
18. Evaluation	Extensions can be easily supported by new XML elements.		
Adoption Issues	3 (Strong industrial support)		
19. Research preference	1 (A lack of sophisticated theoretical and semantic basis)		
20. Industry preference	3 (Strong interest in the industrial community)		
21. Standards impact	3 (Contributions to ebXML and the related efforts)		

7 FIPA-SL

7.1 Basis of the Review

The basis of this review is the FIPA standard specification for FIPA-SL [FIPA00061].

Since there are no commercially supported implementations of reasoning mechanisms for FIPA-SL, it is difficult to evaluate certain of the criteria defined in this document, mainly the migration-path criterion. However, future implementations of FIPA-SL processing tools may provide the means for defining new constructs from previous ones. FIPA-SL provides the means for all requirements related to language Expressivity, but only syntactically. When talking about languages with inference engines, such as Prolog, they may also be supported by the inference mechanism.

Objectively, Prolog allows the syntactic representation of all kinds of expressions which are allowed by FIPA-SL and more. All such expressions would be parsed by the Prolog `read/1` command. However, some of them would not be directly supported by the reasoning mechanism of Prolog.

7.2 Expressivity

7.2.1 Expressivity Test

Expression	Language Representation	Details

"Schrödinger's Cat is alive"	`(forall ?x`
	`  (implies (and`
	`  (owned_by schrödinger ?x)`
	`  (cat ?x))`
	`  (alive ?x)))`
	or
	`(alive (cat :owner "schrödinger"))`	

"Cats are animals"	`(forall ?x (implies` `  (cat ?x) (animal ?x)))`	
"You making the tea"	`(action luis make-tea)`	

"Drinking too much is bad for you"	`(forall ?x (implies`
	`  (drunk_too_much ?x)`
	`  (bad_for ?x)))`	

"All red things"	**`(all ?x (red ?x))`**	

"Any colour a car might have"		Car must be existentially quantified outside of the `iota` operator. If car has a concrete identifier (for example, car123), it becomes possible:
		`(iota ?x`
		`  (colour car123 ?x))`

"All things are hot"	`(forall ?x (hot ?x))`	
"Something is cold"	`(exists ?x (cold ?x))`	
"Herring or Perch"		
"Vodka and Tonic"		
"Not cricket"		
"Success implies Payment"	**`(forall ?x (implies`** **`  (successful ?x)`** **`  (must_pay ?x)))`**	
"Luis has the persistent goal that W"	`(PG luis W)`	
"Steve Believes X"	`(B steve X)`	
"Jonathan Desires Y"		This could be written as a

		functional term, but it would have no pre-defined semantics.
"Matthias Intends Z"	`(I Matthias Z)`	

7.2.2 Summary

As shown in the table, FIPA-SL can represent most of the test statements.

Comment: This table was not complete and had extra test statements which I have removed.

Luis or Steve, can you please fill in the blanks to support this statement.

Allows modal operators

FIPA-SL provides all the mentioned modal operators, but the U operator does not have a well-defined semantics. From a practical point of view, this drawback is not an important impairment since uncertainty has long been addressed using conceptual and computational tools without clearly defined semantics, such as the confidence factor approach and fuzzy logic.

7.3 Parsing and Processing Tools

EPFL has implemented Java parsers for FIPA-SL with ATOMIK [ATOMIK] which provides support for multiple languages. ADETTI has also implemented a Java Parser for FIPA-SL but it has not yet been tested (in the real sense of the word). Since ADETTI has also implemented a XML-Schema based Java parser for XML it may also be used to parse any future XML definition of FIPA-SL as long as a XML-Schema is defined.

In addition to these, the following FIPA platforms have integrated support for (full) FIPA-SL:

- Agentworks platform,
- April Agent Platform [AAP],
- Comtec Agent Platform, and,
- FIPA-OS.

All other platforms in the EU Agentcities.RTD project support at least FIPA-SL0[11].

7.3.1 Translators

ADETTI has developed a simple translator for the s-expression syntax of FIPA-SL to a possible Prolog syntax of FIPA-SL. The translator has been implemented in Prolog, but it has not been seriously tested. ADETTI is also implementing a Java program to translate a subset of FIPA-SL into SQL.

7.3.2 Reasoners

ADETTI is developing a C++ inference engine for a subset of FIPA-SL which includes first-order logic for finite domains with existential quantification but not with universal quantification.

7.4 Knowledge Acquisition Facilities

There are no knowledge acquisition tools for SL that we know of.

7.5 Language Learning Difficulty

Since SL is a logic-based language, its learning difficulty may be considerable for people without previous knowledge of logic. Moreover, since there is no tool to help edit SL expressions, it is very difficult to debug SL.

- **Agent developers**

  Agent developer will learn how to use SL in concrete examples by comparison with other examples. If an agent developer has some prior knowledge of other logic-based languages, such as KIF, he or she will learn SL more easily. Undergraduate students can learn SL through examples in about two weeks teaching in one of five courses.

[11] Extending FIPA-SL is possible but is not necessarily trivial since it requires variable handling and the parse tree for FIPA-SL is considerably deeper than that of FIPA-SL0.

- **Researchers**

  Researches with previous knowledge of logic will have no difficulty learning SL.

- **Lay people**: It is relatively difficult since it is based on logic.
- **Industry**: Adoption is not easy.

7.6 Migration Paths and Flexibility

SL does not provide any explicit extension mechanism. SL is basically a logic-based language with no provision for knowledge structuring as provided by frame-based or object-oriented languages.

With a logical language, however, the most important consideration is dependent on its usage in one of the following two classes:

1. Using SL in a simple and restricted way where it is not expected that agents can understand arbitrary expressions in the language or have access to a theorem prover.
2. Allowing and encouraging the use of arbitrary expressions in the language and expecting all agents to be able to handle them.

The second approach would lead to a dependency on both SL and very expressive logic-based languages, which cuts down the migration paths considerably. This would appear to apply to all of the expressive languages being considered, such as, FIPA-SL, Prolog and KIF. It should be relatively easy to migrate from FIPA-SL to KIF, although it is not clear that KIF has the same expressiveness as SL.

7.7 Adoption Issues

SL is unlikely to be popular with any community since it is very recent and used only within the FIPA community, which does not like it particularly. SL can, however be accepted by some communities but not by others:

- **Research**: In general, it would be well received by the artificial intelligence community and relatively badly by most agent engineers currently using arbitrary content.
- **Industry**: Is unlikely to be popular since it is tagged as a research language.
- **Standards**: SL is used only in the FIPA Specs. However its use is not mandatory, even according to the FIPA specs.

7.8 Summary

Expressivity			
01. Represents propositions	Yes		
02. Represents actions	Atomic and Composed		
03. Represents open questions	Yes		
04. Allows quantification	Explicit		
05. Allows connectives	AND, OR, NOT, EQUIV, and IMPLIES		
06. Allows action propositions	Done, and Feasible		
07. Allows modal operators	I, B, G, PG, and U		
08. Allows functions	Yes		
Available Processing Tools	**Tool Name**	**License/Price**	**Comments**
09. Parsers	ADETTI, EPFL and many current FIPA platforms	Open Source	ADETTI is developing an FIPA-SL parser for C Programs
10. Translators	ADETTI	Open Source	Prolog, SQL
11. Reasoners	No		
Knowledge Acquisition Facilities	**Tool Name**	**License/Price**	**Comments**
12. Editors	No		
13. Others	No		

Language Learning Difficulty	1 (Very Easy) – 5 (Very Difficult)
14. Agent Developers	2
15. Researchers	2
16. Industry	3
17. Lay people	4
Migration Paths	
18. Evaluation	FIPA-SL does not provide defining mechanisms therefore it cannot be extended.
Adoption Issues	1 (Low Preference) - 3 (High Preference)
19. Research preference	2 (Similar to other modal logics)
20. Industry preference	1
21. Standards impact	2 (If FIPA-SL was used it would have a big impact in the FIPA-SL specification)

8 KIF Review

8.1 Basis of the Review

The basis of this review is the draft proposed American National Standard (dpANS) definition of KIF (NCITS.T2/98-004) [KIF].

8.2 Expressivity

The Knowledge Interchange Format (KIF) is a well know, logic-based language for expressing knowledge. The language has:

- Lisp like syntax,
- A declarative semantics (that is no procedural aspect to the semantics) is claimed to be one of the language features,
- At its most general it can be used to express arbitrary logical sentences, and,
- Can be used to express meta-knowledge.

KIF semantics are based on a conceptualisation of the world in terms of objects and relations. Basic elements in KIF are terms, sentences and definitions (although only sentences and definitions are defined for use as complete standalone statements). Only restricted subsets of KIF are tractable for reasoning, such as SKIF which is equivalent to Horn clause logic.

8.2.1 Expressivity Test

Expression	Language Representation	Details

"Schrödinger's Cat is alive"	`(holds true`
	`  (is-alive`
	`  schrödinger-cat))`	

"Cats are animals"	`(forall (?x Cat)`
	`  (holds true`
	`  (is-animal ?x)))`	

"You making the tea"	`(make-tea you)`	There are no semantics for actions; this is just a functional term. One could also define the notion of an action expression. Note also that this is not a fully formed KIF sentence or form, it is only a term.

"Drinking too much is bad for you"	`(forall ?x`
	`  (=> (drink-excess ?x)`
	`  (poor-health ?x)))`	

"All red things"	`(?x red)`
	`(get-all ?x`
	`  (is-red ?x))`
		The first two both work, but they are not fully formed KIF sentences or forms. Note that there is no special defined term for `all`, `iota` and `any`.

"Any colour a car might have"	`(?x allowed-car-colour)` `(is-car-colour ?x)`	Similar to the previous one – very weak well.
"All things are hot"	`(forall ?x` `  (holds true (is-hot ?x)))`	There are multiple ways of doing this – this is longhand.
"Something is cold"	`(exists ?x` `  (holds true (is-cold ?x)))`	There are multiple ways of doing this – this is longhand.
"Herring or Perch"	`(or herring perch)`	
"Vodka and Tonic"	`(and vodka tonic)`	
"Not cricket"	`(not cricket)`	

"Success implies Payment"	`(=> success payment)`
	`(forall ?x`
	`  (=> (successful ?x)`
	`  (payment ?x)))`
		There are multiple ways of doing this.

"Luis has the persistent goal that W"	`(hold true (has-pg W luis))`	This is an example of using such a defined function

"Steve Believes X"	`(hold true`
	`  (has-belief X steve))`
	`(believes steve`
	`  '(material moon stilton))`
		See Persistent Goals. Note also the second example which uses a quote to escape another KIF sentence.

"Jonathan Desires Y"	`(hold true` `  (has-desire Y jonathan))`	See Persistent Goals.
"Matthias Intends Z"	`(hold true` `  (has-intent Z matthias))`	See Persistent Goals.
"Matthias Intends Z"	`(hold true` `  (had-intent Z matthias))`	See Persistent Goals.

8.2.2 Summary

KIF is very expressive and it also has a number of features which are not found in all logical-based languages:

- **Quoting**: The quoted section in `(believes steve '(likes jonathan KIF))` is treated at another denotational level.
- **Definitions**: As well as sentences, schemas such as objects, relations, functions, etc. can be defined and hence KIF can be used as its own meta-language and be used to define itself or any other ontology (the most common use of KIF is in building knowledge bases).
- **Flexible syntax**: both prefix and infix forms
- **Flexible expression**: there are quite a number of ways of saying most things (as in most logical languages.

Anything that cannot be expressed using built-in language constructs can be defined in an ontology and used as a functional term, an object, etc., but the semantics must the also be defined. KIF appears to lack built-in support:

- **Referential expressions**: Such as `iota`, `any` and `forall` in FIPA-SL which means that it is not possible to directly refer to a number of objects.
- **Notion of action**: There is no semantics or standardised language support for actions.
- **Notions of belief, intension and persistent goal**: There are no semantics for modal operators.

8.3 Parsing and Processing Tools

While it appears that a large number of people who have used KIF for various knowledge engineering applications there are very few publicly available resources. Furthermore, the resources that are available appear to be rather out of data an under maintained.

Name	Details
MKIF [mKIF]	Aims to implement all (or almost all of KIF). The license is not stated.
Java KIF Parser [JKP]	Implements a parser for a limited form of Horn clause logic, named SKIF. Java based. Last updated in March 1997. The license is free, but the details for commercial use are not specified.
Stanford KIF parser [KIFparser]	Flex, Bison and C++ based parser. The license is not stated.
ATOMIK [ATOMIK]	A multi-language library that supports the same KIF subset as JKP, but the KIF aspect has been implemented but not tested. The license is

	LGPL.
Prologic [Prologic]	Generic reasoning systems that can also be applied to KIF which is Lisp-based. The license is not stated.
EPILOG [EPILOG]	A Lisp-based reasoner for SKIF. The license is free for university, non-profit and commercial use.

8.4 Knowledge Acquisition Facilities

KIF has been used in a considerable number of systems although not always in a rigorous or well documented way. Since KIF was primarily designed for knowledge interchange, much of the usage of KIF has been in the knowledge engineering community and the best-known example of its use is Ontolingua [Ontolingua], which uses an extended KIF syntax as its generic knowledge format.

It should therefore be possible to find bridges between KIF and various knowledge tools, such as OKBC based tools and knowledge bases.

8.5 Language Learning Difficulty

This is difficult to gauge since the KIF specification [KIF] contains almost no examples and examples that do exist are for more advanced parts of the language. There are other examples, such as [SOWA] and in the FIPA-KIF Specification [FIPA-KIF].

- **Positive points**: Syntax is relatively readable but may hide complexity (differences between functional terms, relational sentences etc.), relationship with conceptual graphs (see [SOWA]) for visualisation.
- **Negative points**: The language is very deep and people without a strong logic background are likely to find it quite challenging. There are also a lot of different ways of saying the same thing – making for potentially complex system building. There are no "simplified subsets" that have been standardised (there are two subsets for which parsers exist which may help).

Evaluation by groups:

- **Agent developers**: Possible, but difficult for those without a strong logic background, particularly given the lack of good documentation.
- **Researchers**: Many will already have had some experience with KIF and it is a well-known language. However, it is still likely to be challenging for those without a strong logic background.
- **Industry**: Of the logical language choices, KIF is probably known to an extent that it is not completely new but it would be unrealistic to expect industry adoption without significant tool support to hide much of the logic beneath.
- **Lay people**: KIF is more readable that RDF, for example, but it may be intimidating and constructing KIF sentences would likely be more challenging. Conceptual graphs may help here if they can be used to show the meanings of KIF sentences, but the mapping is only partial.

8.6 Migration Paths and Flexibility

As a logic-based language KIF, is very flexible and, in principle, aspects of the language which are required but not present could easily be defined and added. KIF has an underlying object/relation metaphor which may help in migrating to object oriented languages if desired

With a logical language, however, the most important consideration is dependent on its usage in one of the following two classes:

3. Using KIF in a simple and restricted way where it is not expected that agents can understand arbitrary expressions in the language or have access to a theorem prover.
4. Allowing and encouraging the use of arbitrary expressions in the language and expecting all agents to be able to handle them.

The second approach would lead to a dependency on both KIF and very expressive logic-based languages, which cuts down the migration paths considerably. This would appear to apply to all of the

expressive languages being considered, such as, FIPA-SL, Prolog and KIF, and it should be relatively easy to migrate between KIF and FIPA-SL.

8.7 Adoption Issues

As with the other languages, KIF is likely to be popular with some communities and unpopular with others:

- **Research**: In general, it would be well received by the knowledge engineering community, relatively well by the description logic community, relatively badly by the FIPA community (due to its similarity to but difference from FIPA-SL) and relatively badly by most agent engineers currently using arbitrary content.
- **Industry**: Is unlikely to be popular since it is tagged as a research language.
- **Standards**: Whilst there is a KIF standardisation process underway its status is unclear.

8.8 Summary

The following table presents the list of all attributes that must be considered in the content language evaluation.

Expressivity			
01. Represents propositions	Yes		
02. Represents actions	No (must be defined)		
03. Represents open questions	Yes/No		
04. Allows quantification	Explicit		
05. Allows connectives	AND, OR, NOT, EQUIV, IMPLIES		
06. Allows action propositions	No (must be defined)		
07. Allows modal operators	No (must be defined)		
08. Allows functions	Yes		
Available Processing Tools	**Tool Name**	**License/Price**	**Comments**
09. Parsers	MKIF, JKP, ATOMIK, Stanford Parser	Free, no license specified or LPGL	None of these parsers appear to be very mature, the first three appear to have fallen into disuse (not maintained) and they all address (potentially different) subsets of KIF
10. Translators	None but some must exist for various KR applications		
11. Reasoners	Prologic and EPILOG	Free for non-commercial use, uncertain for commercial use.	Both from Stanford, both LISP based, both do not appear to be maintained

Knowledge Acquisition Facilities	Tool Name	License/Price	Comments
12. Editors	None found but may exist in the context of knowledge engineering projects		
13. Others			
Language Learning Difficulty	1 (Very Easy) – 5 (Very Difficult)		
14. Agent Developers	2		
15. Researchers	2		
16. Industry	3		
17. Lay people	4		
Migration Paths			
18. Evaluation	Very limited if full expressivity is allowed and reliance on theorem provers develops, reasonable migration paths if usage is limited to simple subsets		
Adoption Issues	1 (Low Preference) - 3 (High Preference)		
19. Research preference	2.5 (It depends strongly on the community, people are likely to appreciate the flexibility but some communities may prefer other similar languages, for example FIPA-SL)		
20. Industry preference	2 (KIF is one of the better-known logic-based languages but still unlikely to be popular with industry)		
21. Standards impact	2 (The fact that KIF is well known plays off against the fact that it seems to be stuck in an inaccessible standardisation process)		

9 Prolog Review

9.1 Basis of the Review

The Review of Prolog as a possible content language was based on [Bratko, 2001].

9.2 Expressivity

9.2.1 Expressivity Test

Expression	Language Representation	Details

"Schrödinger's Cat is alive"	`owns ('schrödinger, X), cat (X), alive (X)`
	or
	`alive (cat_owned_by ('Shrodinger'))`
		Functions are not evaluated in Prolog.

"Cats are animals"	`animal (X):- cat (X)`	This kind of sentence can be stated, used for inference but not easily inferred.
"You making the tea"	`action (luis, make-tea)`	The semantics is given by the client program.
"Drinking too much is bad for you"	`bad_for (X):- drink_too_much (X)`	Of course Prolog has some difficulty representing such lies.
"All red things"	`findall (X, red (X), RedThings)` `hot (X):- red (X)`	

"Any colour a car might have"	`colour (car123, Y)`
	or
	`findall (Y, colour (car123, Y), L)`
		A specific car, say, `car123`.

"All things are hot"	`hot (X):-thing (X)`	Can be stated, can be used for inference, but cannot easily be inferred.
"Something is cold"	`cold (X)`	If used in a question. If this is a statement, this means everything is cold
"Younger than 8 or older than 60"	`age (X, A), (A > 60; A < 8)`	In standard Prolog, A must be instantiated before it is tested.
"The desired movie should be romantic and the cinema should at walk distance"	`movie (X), type (X, romantic), cinema (X, Z), near (Z)`	
"The transport should not be private"	`transport (X), \+ type (X, private)`	If this is a statement, it means X is a transport that is not private. If this is a query, it is asking for some transport that is not public.
"Success implies Payment"	`must_pay (X) :- successful (X)`	
"Luis has the persistent goal that W"	`pg (luis, W)`	Can be parsed, expressed and represented. If inference is needed, Prolog must be extended.
"Steve Believes X"	`bel (steve, X)`	Can be parsed, expressed and represented. If inference is needed, Prolog must be extended.
"Jonathan Desires Y"	`desire (jonathan, Y)`	Can be parsed, expressed and represented. If inference is

		needed, Prolog must be extended.

"Matthias Intends Z"	`current_time (T1),`
	`exists (T2, time (T2,`
	`  intends (Matthias, Z))),`
	`T2 < T1`
		Can be parsed, expressed and represented. If inference is needed, Prolog must be extended. `current_time` must be defined

9.2.2 Summary

Represents open questions

Prolog has the capability to express open questions through its meta-logical operators: `findall`, `bagof` and `setof`. There is no Prolog operator that is the equivalent of the FIPA-SL `iota` operator, but it can easily be defined as follows:

```
iota (T, P, X) :-
  findall (T, P, [X])
```

Allows quantification

Prolog does not provide explicit quantification, but even without any extension, Prolog has the means to implicitly represent quantification.

In questions, by default, all variables are implicitly existentially quantified; double negation may be used to represent universal quantification.

In a statement, all variables are implicitly universally quantified. The effects of existentially quantified variables can be achieved in statements only by the predicate definition mechanism, which is a limitation of the language.

Allows connectives

Prolog has the standard conjunction and disjunction operators.

Implication is also possible but it is limited to expressions in which the consequent must be a positive atomic proposition.

Most Prolog implementations have only negation as failure but not logical negation. Negation as failure is everything that is not known to be true is assumed to be false. Some implementations of Prolog (especially those that are based on the Edinburgh syntax) also have logical negation.

Prolog does not provide equivalence.

All connectives may be fully implemented in Prolog by extending the language, which is often the case in AI applications.

Allows action propositions

Action operators do not belong to the language, but an explicit version (which cannot be inferred) of action operators may be easily used.

Allows modal operators

Modal operators do not belong to the language. An explicit version (which cannot to be inferred) of modal operators may be easily used. Reasoning may also be implemented with modal operators, especially modal operators that are closed under the reasoning capabilities of the agent.

Allows functions

Syntactically, Prolog allows functions but it does not evaluate functions with a few exceptions, such as arithmetic operators. The easiest way of circumventing this problem is by using predicates instead of functions, which may result in lengthier expressions. Another way to address the problem is to extend the language with the capability to evaluating functions.

9.3 Parsing and Processing Tools

There are several implementations of Prolog. Some of them are available for the Windows operating system and others for the Linux and UNIX operating systems. In general, Prolog may be integrated with other languages, especially with C/C++. Some Prolog implementations, such as the Win-Prolog and the SICStus Prolog may also be integrated with Java programs.

There are also several Prolog implementations (such as JavaLog) that are implemented as Java classes therefore they may easily be used in Java programs. Unfortunately these implementations are not well documented nor well supported.

Some Prolog systems may also access relational databases through ODBC and the Internet through special purpose libraries, for example, Win-Prolog.

Name	Details
Win-Prolog [WinProlog]	Prolog for the Windows operating system which can be embedded in Java, C/C++ and VB programs
SICStus Prolog [SICStus]	Prolog for Windows, Linux and UNIX operating systems which can be embedded in Java and C/C++ programs
Amzi Prolog [Amzi]	Prolog for Windows, Linux and UNIX operating systems which can be embedded in Java and there is an IDL compiler for Amzi Prolog
GNU Prolog [GNU-Prolog]	Open source Prolog for the Linux operating system which can be integrated with C/C++ programs
Visual Prolog [Visual-Prolog]	Prolog for Windows, OS/2, and SCO UNIX operating systems which an be integrated with C/C++ programs and can also call Windows library files
SWI Prolog [SWI-Prolog]	Prolog for Windows, Linux, and Solaris UNIX which can be integrated with Java and C/C++ and there is an IDL compiler for SWI Prolog
JavaLog [JavaLog]	Prolog implemented as a Java class which has little support and documentation
Bin Prolog [Bin-Prolog]	Prolog for Windows, Linux and Solaris UNIX operating system which generates C/C++ code that can be embedded within Java and C/C++ programs. It is a multi-threaded programming environment with a AI tools such as blackboards, etc.
Quintus Prolog [Quintus]	Prolog for Windows and UNIX operating systems where programs may be called from C/C++ programs and may call routines in other languages, but only from Java programs under the Windows operating system

9.3.1 Parsers

It is easy to parse Prolog expressions since a single `read` command may be used to read and parse any Prolog expression from any stream, including strings, sockets, files, etc. A stream containing a character sequence represented by:

```
forall (X, implies (p (X), q (X))
```

It can be read using the `read/1` predicate and it is automatically converted into a symbolic tree structure.

9.3.2 Translators

It is unknown if there are any commercially available translators from Prolog to other languages or from other languages to Prolog. However, Prolog has a built-in formalism to write grammars called DCG which may be used to express a variety of grammars (including context dependent grammars) and to create parsers for those grammars. ADETTI has built a simple translator from FIPA-SL to Prolog.

9.3.3 Reasoners

A Prolog interpreter is a reasoning mechanism. If the Prolog reasoning mechanism is not appropriate for the problem, it is a relatively simple matter to define other kinds of reasoning mechanisms on top of Prolog. Both logic-based reasoning mechanisms and non-logic reasoners such as planners and learning algorithms may also be implemented.

9.4 Knowledge Acquisition Facilities

There are many Prolog implementations. Some of them have complex sophisticated development environments. Others are very simple tools often with no more than a prompt-based interpreter. Even in the simplest case, the interpreter can also be used to perform syntax validity tests.

9.5 Language Learning Difficulty

In terms of learning difficulty, no studies are known but by experience of some of the authors, Prolog can easily be learned by people without previous knowledge of other programming languages.

Although Prolog is a logic-based language, its learning difficulty is not as much as that of SL or KIF, because its syntax is simper and also because it is a programming language and therefore it is easy to test the effects of designed message contents when interacting with a program.

- **Agent developers**

  Agent developer will learn how to use Prolog in concrete examples by comparison with other examples, and also trying their guesses with simple example programs playing the role of the receiver. If an agent developer has some prior knowledge of other logic-based languages, such as KIF, he or she will learn Prolog more easily. Undergraduate students without any prior knowledge of logic can fully learn how to use Prolog in half of one of five courses.

- **Researchers**

  Researches with previous knowledge of logic will have no difficulty learning Prolog.

- **Lay people**: It is relatively difficult since it is based on logic. But it is easier than other logic-based languages because its syntax is simpler and its use may be tested with an interpreter

Industry: Adoption is easy, especially in community niches where Prolog is relatively known such as in the UK, in France and in Japan.

9.6 Migration Paths and Flexibility

Being a programming language, Prolog can easily be extended. Since besides being a programming language, Prolog has strong symbolic processing capabilities, the possibility to extend it is far more powerful than that of competitors.

Since Prolog has quoting mechanism, it can be used with embedded expressions in other languages, but those will not be interpreted by the original Prolog unless it is extended.

Prolog can also be embedded in programs written other programming languages such as C and Java.

9.7 Adoption Issues

Prolog is likely to become popular if the project adopted it and increase its dissemination. However there are also research communities that offer strong resistance to the use of Prolog.:

- **Research**: In general, it would be well received by part of the artificial intelligence community and relatively well by most agent engineers currently using arbitrary content.
- **Industry**: Is likely to become popular if it were adopted by the project, since it can also be used as a programming language.
- **Standards**: There is an ISO Prolog standard but, in fact, it is not much used. It is not difficult that FIPA would accept a Prolog Content Language in its Content Language Library since Prolog provides the means to represent all content types required by FIPA ACL.

9.8 Summary

Expressivity			
01. Represents propositions	Yes		
02. Represents actions	Atomic and composed		
03. Represents open questions	Yes		
04. Allows quantification	Implicit		
05. Allows connectives	AND, OR, NOT, EQUIV, IMPLIES		
06. Allows action propositions	Done, Feasible		
07. Allows modal operators	I, B, G, PG, U		
08. Allows functions	Yes		
Available Processing Tools	**Tool Name**	**License/Price**	**Comments**
09. Parsers	Any Prolog	Variable	
10. Translators	No		
11. Reasoners	Any Prolog		
Knowledge Acquisition Facilities	**Tool Name**	**License/Price**	**Comments**
12. Editors	Not usually	Variable	
13. Others	Yes		
Language Learning Difficulty	1 (Very Easy) – 5 (Very Difficult)		
14. Agent Developers	2		
15. Researchers	2		
16. Industry	3		
17. Lay people	3		
Migration Paths			
18. Evaluation	Prolog is very easy to extend, therefore new capabilities may easily be added if needed.		
Adoption Issues	1 (Low Preference) - 3 (High Preference)		
19. Research preference	2 (Certain communities would really appreciate the idea but some others would not)		
20. Industry preference	1		
21. Standards impact	2 (It would be very easy to create a specification of Prolog as a content language o its own or as a concrete syntax for ACL and also for FIPA-SL)		

10 Comparative Evaluation and Conclusions

This section briefly summarises the conclusions from the review for each of the potential languages and indicates the final decisions taken in the EU Agentcities.RTD project as initial content languages for the Agentcities Network.

10.1 Recommendations for DAML+OIL

1. DAML+OIL might not the correct tool to use as a generic content language because of its restricted expressivity. However, DAML+OIL may prove useful in the following areas:
   - Direct communication about ontologies between agents, perhaps ensuring that content languages we chose can embed DAML+OIL.
   - Definition of language elements for new or existing content languages (meta-level) which might then make it possible to apply DAML+OIL tools to statements in the chosen content language.
2. As a general communication language it is impossible to (cleanly) express many important concepts, such as action expressions, object references, compound logical and modal statements.

10.2 Recommendations for ebXML

1. ebXML is not a traditional knowledge representation framework and is therefore not as expressive as the typical knowledge representation languages like KIF, FIPA-SL or Prolog. The focus of ebXML is on the direct and explicit representation of information items needed in business collaborations and coordination. Instead of having a meta-language that can represent everything, the strategy of ebXML is to have an extensible framework that supports core elements, which can be easily extended to meet new requirements. The relation to traditional EDI frameworks and the support from the industrial community are the other important characteristics.
2. ebXML is not suitable as a generic content language for agent communication.
3. A possible content language and ontology model should at least integrate the key features and functionalities of ebXML to offer suitable support for e-Commerce collaborations.
4. The possibility of direct translations to ebXML representations can be advantageous in easing the integration with existing and future e-Commerce platforms and in increasing the acceptance of solutions from Agentcities by the industrial community.

10.3 Recommendations for Prolog

1. Prolog is less expressive than FIPA-SL in the following sense: although we can write and parse any of the FIPA-SL expressions in Prolog, some of them would not mean a thing for the Prolog inference engine.
2. Prolog has commercial support for parsing, syntax checking and inference. However, this support is only effective if the agent implementation language is also Prolog. To use Prolog as a content language for Java agents, there would not be any advantage in relation to using FIPA-SL or other languages, except for parsing.
3. To integrate Prolog code with Java code, two approaches can be followed. The first is to write two different programs that communicate via sockets, or the second is to embed Prolog predicates in a Java program. The first is always possible, but the second alternative can only be used with a few Prolog implementations, some of which are not free.
4. Prolog is easy to expand with new capabilities than it is to extend any other language, such as Java or C/C++.
5. Prolog is relatively easy to learn but it is not a current trend.

10.4 Recommendations for FIPA-SL

1. FIPA-SL is more expressive than the other languages: The expressive power of FIPA-SL is well founded in semantic terms (with a few exceptions and some clarifications), which ensures that

some inference properties could be implemented. This is not the case for other candidates such as Prolog or DAML+OIL. However, since no reasoning support exists for FIPA-SL, the expressive power of the language is not an advantage in relation to other candidate content languages that allow expression the same constructs.

2. FIPA-SL parsers have been built by other EU Agentcities.RTD project partners, which make them free and easily adaptable to the needs of the project.
3. FIPA-SL does not provide the means for extension.
4. FIPA-SL is relatively easy to learn but it is not a current trend.

10.5 Recommendations for KIF

- Even though KIF is primarily designed for knowledge bases it can express most of the things in the test set (and much more). There are some expressions it has no direct support for modal operators, actions and referential expressions.
- KIF has been soundly tested for various knowledge engineering applications.
- Between FIPA-SL and KIF, KIF is likely to have an advantage over FIPA-SL since it is more widely know. With reference to the other languages, it is difficult to say since they are differing functionality.
- From the parsers available, most do not seem to be maintained, do not address different subsets of the language, and, have (to our knowledge) never been tested together.
- Adopting KIF in the EU Agentcities.RTD project would require agreed language extensions for action expressions (syntactic and semantic), agreement extensions for referential expressions (syntactic and semantic), agreement on a potential subset of the languages to use, development effort on one or more parsers for that subset, and, integration of these parsers into the existing agent platforms.

10.6 Final Recommendations

Based of on the evaluations given in this document and experiences of the EU Agentcities.RTD project partners the final choice made was as follows:

- Individual services can be built in either FIPA-SL or KIF.
- Work is to be carried out to establish necessary and useful subsets of both languages for use in the project; while the full power of each language does not seem to be required it is also not clear a-priori what is necessary for the target applications.
- Work is to be carried out to map subsets of the two languages to one another to facilitate interoperable service development.

11 References

[AAP] *April Agent Platform*
http://sf.us.agentcities.net/aap/

[Amzi] *AMZI Prolog and Logic Server* http://www.amzi.com/

[Algae] *RDF Query Language*
http://www.w3.org/1999/02/26-modules/User/Algae-HOWTO.html

[ATOMIK] *Language and Ontology Toolkit*
http://liawww.epfl.ch/ATOMIK/

[AUML] *Agent-UML Modelling Language* http://www.auml.org.

[Bratko, 2001] Ivan Bratko. 2001. "Prolog Programming for Artificial Intelligence. Third Edition".

[Bin-Prolog] *Bin-Prolog Toolkit* http://www.binnetcorp.com/BinProlog/

[DAML+OIL] *DAML+OIL Language Specification*, March 2001
http://www.daml.org/2001/03/daml+oil.daml
DAML+OIL Example Ontology, March 2001
http://www.daml.org/2001/03/daml+oil-ex.daml
DAML+OIL Walk-thru, March 2001
http://www.daml.org/2001/03/daml+oil-walkthru.html

[DAML-API] *DAML API*
http://grcinet.grci.com/maria/www/codipsite/Tools/Components.html

[ebXML] *Technical Architecture Specification*, version 1.0.4
Business Process Specification Schema, version 1.01
Registry Service Specification, version 1.0
Collaboration-Protocol Profile and Agreement Specification, version 1.0

[EPILOG] *EPILOG Inference Package*, Stanford University, 1995
http://logic.stanford.edu/sharing/programs/epilog/

[FIPA00008] *FIPA SL Content Language Specification*, FIPA, 2000.
http://www.fipa.org/specs/fipa00008/

[FIPA00061] *FIPA ACL Message Structure Specification*, FIPA, 2000.
http://www.fipa.org/specs/fipa00061/

[GNU-Prolog] *BNU-Prolog Toolkit* http://org.gnu.de/software/prolog/prolog.html
http://pauillac.inria.fr/~diaz/gnu-prolog/

[KIF] *Knowledge Interchange Format Draft ANSI Proposal,* dpANS NCITS.T2/98-004
http://logic.stanford.edu/kif/dpans.html

[KQML] *Specification of the KQML Agent-Communication Language -- plus example agent policies and Architectures*, Tim Finin et. Al. The DARPA Knowledge Sharing Initiative External Interfaces Working Group Technical Report 1993. URL:
http://www.cs.umbc.edu/kqml/papers/kqmlspec.ps

[JavaLog] *JavaLog Prolog Toolkit* http://www.exa.unicen.edu.ar/~azunino/javalog.html

[Jena] *RDF/DAML+OIL Parser*
http://www.hpl.hp.com/semweb/jena-top.html

[JKP] *Java KIF Parser*, X. Luan
http://www.csee.umbc.edu/kse/kif/jkp/

[KIFparser] *Stanford KIF Parser*. Gregory Olsen
http://piano.stanford.edu/concur/software/kifparser.html

[FIPA-KIF] *FIPA KIF Content Language Specification*, Foundation for Intelligent Physical Agents, 2000
http://fipa.org/specs/fipa00010/

[mKIF] *mKIF Parser*, Mariusz Nowostawski
http://marni.otago.ac.nz/mKIF/

[Ontolingua] *Ontolingua Knowledge Server*, Stanford University
http://www-ksl-svc.stanford.edu:5915/

[Prologic] *Prologic Reasoning Systems*, Standford University
http://logic.stanford.edu/sharing/programs/prologic/

[Quintus] *Quintus Prolog Toolkit* http://www.sics.se/quintus/

[Redland] *RDF Framework*
http://www.redland.opensource.ac.uk/

[Sesame] *RDF Query Language*
http://sesame.aidministrator.nl/

[SICStus] *Sictus Prolog toolkit* http://www.sics.se/isl/sicstus.html

[SOWA] *Conceptual Graph Examples*
http://users.bestweb.net/~sowa/cg/cgexampw.htm

[SWI-Prolog] *SWI Prolog Toolkit* http://www.swi.psy.uva.nl/projects/SWI-Prolog/
http://gollem.swi.psy.uva.nl/twiki/pl/bin/view/Foreign/JavaInterface

[Visual-Prolog] *Visual Prolog Toolkit* http://www.visual-prolog.com/

[Wilbur] *RDF Toolkit*
http://wilbur-rdf.sourceforge.net/

[WinProlog] *WinProlog Toolkit* http://www.lpa.co.uk/